



Grid Programming in Python with pyGlobus

Keith R. Jackson

*Distributed Systems Department
Lawrence Berkeley National Lab*



Overview

- Python CoG Kit Overview / Goals
- Why Python?
- Status
- Module Overview
 - Job Submission
 - Data Transfer
 - GridFTP
 - GassCopy
 - Remote File IO
 - Secure Network IO
 - GSI Soap
- OGSI Support
- Wrapping Legacy Codes
- Future Plans
- Contacts & Acknowledgements



Python CoG Kit

- The Python CoG Kit provides a mapping between Python and the Globus Toolkit. It extends the use of Globus by enabling access to advanced Python features such as events and objects for Grid programming.
- Hides much of the complexity of Grid programming behind simple object-oriented interfaces.
- The Python CoG Kit is implemented as a series of Python extension modules that wrap the Globus C code.
- Uses SWIG (<http://www.swig.org>) to help generate the interfaces.



Why Python?

- Easy to learn/read high-level scripting language
 - Very little syntax
- A large collection of modules to support common operations, e.g., networking, http, smtp, ldap, XML, Web Services, etc.
- Excellent for “gluing” together existing codes
 - Many automated tools for interfacing with C/C++/Fortran
- Support for platform independent GUI components
- Runs on all popular OS's, e.g., UNIX, Win32, MacOS, etc.
- Support for Grid programming with pyGlobus, PyNWS, etc.

Python CoG Kit Goals



- Supports rapid development of high-level Grid services, while providing access to the underlying performance of the Globus toolkit.
- Develop a common set of reusable components in Python for accessing Grid services.
- Focus on supporting the development of Science Portals, Problem Solving Environments, and science applications that access Grid resources.



Python CoG Kit Basics

- Everything is contained in the pyGlobus package.
- The low-level C-wrappers are all named modulec, e.g., ftpClientc, gramClientc
 - Shouldn't need to access these directly, instead use the Python wrappers (ftpClient, gramClient)
- Exceptions are used to handle error conditions
 - `pyGlobus.util.GlobusException` is the base class for all pyGlobus exceptions. It extends the base Python `exceptions.Exception` class
- The Python wrappers classes manage the underlying memory & module activation/deactivation



Status: Python CoG Kit

- Basic services are provided accessing:
 - Security (security)
 - Remote job submission and monitoring (gramClient)
 - Secure high-performance network IO (io)
 - Protocol independent data transfers (gassCopy)
 - High performance Grid FTP transfers (ftpClient)
 - Support for building Grid FTP servers (ftpControl)
 - Remote file IO (gassFile)
 - GASS Cache management (gassCache)
 - Replica management (replicaManagement)
 - Replica Catalog (replicaCatalog)



gramClient Example

```
from pyGlobus.gramClient import *
from threading import *
cond = 0
rm = "host.lbl.gov"
rsl = "&(executable=/bin/date)"

def done(cv,contact,state,err):
    global cond
    if state == JOB_STATE_FAILED:
        print "Job failed"
    elif state == JOB_STATE_DONE:
        print "Job is done"
    else:
        print "ERROR: ", err
    cv.acquire()
    cond = 1
    cv.notify()
    cv.release()

def main(rm, rsl):
    condV = Condition(Lock())
    try:
        gC = GramClient()
        cbContact =
            gC.set_callback(done, condV)
        jobContact =
            gC.submit_request(rm, rsl,
                               JOB_STATE_ALL, cbContact)
        while cond == 0:
            condV.acquire()
            condV.wait()
            condV.release()
        gC.remove_callback(cbContact)
    except GramClientException, ex:
        print ex
```



C Job Submission Example

```
callback_func(void *user_arg, char *job_contact,
             int state, int errorcode)
{
    globus_i_globusrun_gram_monitor_t *monitor;
    monitor = (globus_i_globusrun_gram_monitor_t *) user_arg;
    globus_mutex_lock(&monitor->mutex);
    monitor->job_state = state;
    switch(state)
    {
        case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
        {
            globus_i_globusrun_gram_monitor_t *monitor;
            monitor = (globus_i_globusrun_gram_monitor_t *) user_arg;
            globus_mutex_lock(&monitor->mutex);
            monitor->job_state = state;
            switch(state)
            {
                case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
                    if(monitor->verbose)
                    {
                        globus_libc_printf("GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED\n");
                    }
                    monitor->done = GLOBUS_TRUE;
                    break;
                case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
                    if(monitor->verbose)
                    {
                        globus_libc_printf("GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE\n");
                    }
                    monitor->done = GLOBUS_TRUE;
                    break;
            }
            globus_cond_signal(&monitor->cond);
            globus_mutex_unlock(&monitor->mutex);
        }
    }
}

globus_l_globusrun_gramrun(char * request_string,
                           unsigned long options,
                           char *rm_contact)
{
    char *callback_contact = GLOBUS_NULL;
    char *job_contact = GLOBUS_NULL;
    globus_i_globusrun_gram_monitor_t monitor;
    int err;
    monitor.done = GLOBUS_FALSE;
    monitor.verbose=verbose;
    globus_mutex_init(&monitor.mutex, GLOBUS_NULL);
    globus_cond_init(&monitor.cond, GLOBUS_NULL);

    err = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    if(err != GLOBUS_SUCCESS)
    {
        ...
    }
    err = globus_gram_client_callback_allow(
        globus_l_globusrun_gram_callback_func,
        (void *) &monitor,
        &callback_contact);
    if(err != GLOBUS_SUCCESS)
    {
        ...
    }
    err = globus_gram_client_job_request(rm_contact,
                                         request_string, GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
                                         callback_contact, &job_contact);
    if(err != GLOBUS_SUCCESS)
    {
        ...
    }
    globus_mutex_lock(&monitor.mutex);
    while(!monitor.done)  {
        globus_cond_wait(&monitor.cond, &monitor.mutex);
    }
    globus_mutex_unlock(&monitor.mutex);
    globus_gram_client_callback_disallow(callback_contact);
    globus_free(callback_contact);
}
```



Data Transfer

- Provides both protocol dependent and protocol independent transfer mechanisms
 - Directly use the Grid FTP protocol to transfer the data.
 - Uses GSI for mutual authentication.
 - Supports partial file transfers and restarts.
 - Supports tunable network parameters.
 - Provides extensions to standard FTP protocol to support striped and parallel data transfer.
 - Use the gassCopy module for protocol independent transfers.
 - It currently supports the ftp, gsiftp, http, and https protocols.



ftpClient Example

```
from pyGlobus import ftpClient
handleAttr = ftpClient.FtpClientHandleAttr()
opAttr = ftpClient.FtpClientOperationAttr()
marker = ftpClientRestartMarker.FtpClientRestartMarker()
fClient = ftpClient.FtpClient(handleAttr)
fClient.get(url, opAttr, marker, done_func, condV)
handle = fClient.register_read(buf, data_func, 0)

def data_func(cv, handle, buffer, bufHandle, bufLen, offset,
             eof, error):
    g_dest.write(buffer)
    if not eof:
        try:
            handle = fClient.register_read(g_buffer, data_func, 0)
        except Exception, e:
```



Performance Options in ftpClient

- Setting tcpbuffer size

```
from pyGlobus import ftpControl  
battr = ftpControl.TcpBuffer()  
battr.set_fixed(64*1024)
```

Or

```
battr.set_automatic(16*1024, 8*1024, 64*1024)  
opAttr.set_tcp_buffer(battr)
```

- Setting parallelism

```
para = ftpControl.Parallelism()  
para.set_mode(ftpControl.PARALLELISM_FIXED)  
para.set_size(3)  
opAttr.set_parallelism(para)
```



Using `ftpClient` plugins

- Debug Plugin

```
from pyGlobus import ftpClient
f = open("/tmp/out", "w+")
debugPlugin = ftpClient.DebugPlugin(f, "foobar")
handleAttr.add_plugin(debugPlugin)
```

- RestartMarker Plugin

```
restartPlugin =
    ftpClient.RestartMarkerPlugin(beginCB,
                                   markerCB, doneCB, None)
```



gassCopy Example

```
from pyGlobus import gassCopy
```

```
srcAttr      = gassCopy.GassCopyAttr()
handleAttr   = gassCopy.GassCopyHandleAttr()
destAttr     = gassCopy.GassCopyAttr()
ftpSrcAttr   = gassCopy.FtpOperationAttr()
ftpDestAttr  = gassCopy.FtpOperationAttr()
srcAttr.set_ftp(ftpSrcAttr)
destAttr.set_ftp(ftpDestAttr)
copy = gassCopy.GassCopy(handleAttr)
copy.copy_url_to_url(srcUrl, srcAttr, destUrl, destAttr)
```



Remote File IO

- Supports reading and writing remote files from http, https, ftp, gsiftp servers.
 - Can be opened either as normal Python file objects or as int file descriptors suitable for use with the os module.

```
from pyGlobus import gassFile  
gassFile = gassFile.GassFile()  
f = gassFile.fopen("gsiftp://foo.bar.com/file", "r")  
lines = f.readlines()  
gassFile.fclose(f)
```



Secure High-Performance IO

- Uses the Grid Security Infrastructure to provide authentication.
- Provides access to the underlying network parameters for tuning performance.
- Integrates with the Python networking model



GSI TCP Server Example

```
from pyGlobus import io
attr = io.TCPIOAttr()
attr.set_authentication_mode(
    io.GLOBUS_IO_SECURE_AUTHENTICATION_MODE_GSS_API)
authData = io.AuthData()
authData.set_callback(auth_callback, None)
attr.set_authorization_mode(
    io.GLOBUS_IO_SECURE_AUTHORIZATION_MODE_CALLBACK,
    authData)
attr.set_channel_mode(
    io.GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP)
soc = io.GSITCPSocket()
port = soc.create_listener(attr)
soc.listen()
childSoc = soc.accept(attr)
buf = Buffer.Buffer(size)
bytesRead = childSoc.read(buf, size, size)
```



GSI TCP Client Example

```
from pyGlobus import io
attr = io.TCPIOAttr()
attr.set_authentication_mode(
    io.GLOBUS_IO_SECURE_AUTHENTICATION_MODE_G
SS_AP)
authData = io.AuthData()
attr.set_authorization_mode(
    io.GLOBUS_IO_SECURE_AUTHORIZATION_MODE_SE
LF, authData)
attr.set_channel_mode(
    io.GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRA
P)
soc = io.GSITCPSocket()
soc.connect(host, port, attr)
nBytes = soc.write(str, len(str))
```

Secure MDS Access



- Uses the standard Python ldap module with GSI SASL support.

```
import ldap, ldap.sasl  
l = ldap.initialize("ldap://scott.lbl.gov:2135")  
auth = ldap.sasl.gsigssapi()  
l.sasl_bind_s("", auth)  
res = l.search_s("Mds-Vo-name=DSD, o=Grid",  
    ldap.SCOPE_SUBTREE, "(objectClass*)")  
print res l.unbind()
```



GSISOAP Client Example

- Built on SOAP.py, switching to ZSI
- GSISOAP client example

```
from pyGlobus import GSISOAP, ioc

proxy =
    GSISOAP.SOAPPProxy("https://host.lbl.gov:8081",
    namespace="urn:gtg-Echo")
proxy.channel_mode =
    ioc.GLOBUS_IO_SECURE_CHANNEL_MODE
proxy.delegation_mode =
    ioc.GLOBUS_IO_SECURE_DELEGATION_MODE_NONE
print proxy.echo("spam, spam, spam, eggs, and
spam")
```



GSISOAP Server Example

```
from pyGlobus import GSISOAP, ioc

def echo(s, _SOAPContext):
    cred = _SOAPContext.delegated_cred
    # Do something useful with cred here
    return s

server = GSISOAP.SOAPServer(host.lbl.gov, 8081)
server.channel_mode =
    ioc.GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP
server.delegation_mode =
    ioc.GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY
server.registerFunction(SOAP.MethodSig(echo,
    keywords=0, context=1), "urn:gtg-Echo")
server.serve_forever()
```



pyGlobus Requirements

- Python 2.0+
- Globus 2.0
- Support for dynamic libraries



Open Grid Services Infrastructure

- Develop a full OGSI implementation in Python
 - Planned alpha release of an OGSI client by the middle of September
 - OGSI hosting environment based on WebWare (<http://webware.sourceforge.net/>)
- Dynamic web service invocation framework
 - Similar to WSIF (Web Services Invocation Framework) from IBM for Java
 - <http://www.alphaworks.ibm.com/tech/wsif>
 - Download and parse WSDL document, create request on the fly
 - Support for multiple protocol bindings to WSDL portTypes



Wrapping Legacy Codes

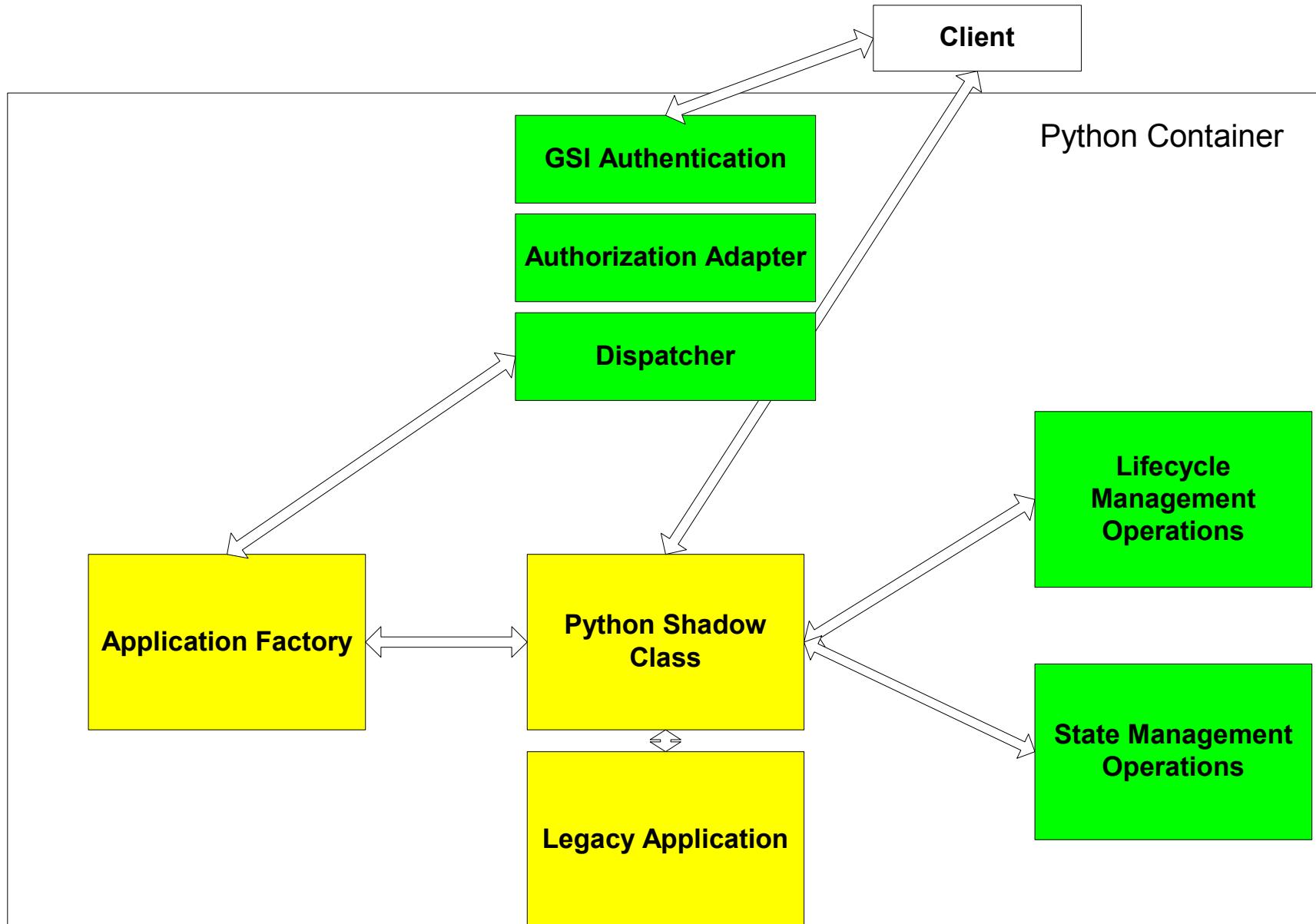
- Many existing codes in multiple languages, e.g., C, C++, Fortran
 - Would like to make these accessible on the Grid
 - Should be accessible from any language
 - Need to integrate standard Grid security mechanisms for authentication
 - Need a standard framework for doing authorization
 - Would like to avoid custom “one off” solutions for each code



Wrapping Legacy Codes (cont.)

- Provide a framework that legacy applications can easily be plugged into
 - Must be easy to add applications written in many languages
 - Use the Python language as the “glue”
- The framework should support:
 - Authentication using standard Grid mechanisms
 - Flexible authorization mechanisms
 - Lifecycle management
 - Including persistent state
- Develop one container, and reuse it for many legacy applications
- Use Web Services protocols to provide language neutral invocation and control
 - Use standard high-performance Grid protocols, e.g., GridFTP, for data transfer

Wrapping Legacy Codes (cont.)



Tools for Automated Interface Generation



- SWIG (Simple Wrapper Interface Generator)
 - Generates interfaces from C/C++
 - Supports the full C++ type system
 - Can be used to generate interfaces for Python, Perl, Tcl, Ruby, Guile, Java, etc.
 - Automatic Python “shadow class” generation
 - <http://www.swig.org/>
- Boost.Python (Boost Python interface generator)
 - Generates interfaces from C++
 - <http://www.boost.org/libs/python/doc/>
- PyFort (Python Fortran connection tool)
 - Generates interfaces from Fortran
 - <http://pyfortran.sourceforge.net/>
- F2PY (Fortran to Python Interface Generator)
 - Generates interfaces from Fortran
 - <http://cens.ioc.ee/projects/f2py2e/>



Future Plans

- Windows support
- Develop OGSI client modules
- Develop an OGSI server framework
 - Based on WebWare
- Explore XML based GUI description tools for the automatic generation of GUI components
 - Should work with both the Java and Python CoG Kits



Contacts / Acknowledgements

- <http://www-itg.lbl.gov/>
- krjackson@lbl.gov
- This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract DE-AC03-76SF00098 with the University of California.